



IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

AF/2176 \$  
2700

IBM application of: Cooper et al.

Serial No.: 09/306,189

Filed: May 6, 1999

For: Method and Apparatus for  
Converting Programs and Source  
Code Files Written in a Programming  
Language to Equivalent Markup  
Language Files

35525

PATENT TRADEMARK OFFICE  
CUSTOMER NUMBER

§  
§  
§  
§  
§  
§

Group Art Unit: 2176

Examiner: Yuan, Almari Romero

Attorney Docket No.: AT9-98-920

Certificate of Mailing Under 37 C.F.R. § 1.8(a)

I hereby certify this correspondence is being deposited with the United States Postal Service as First Class mail in an envelope addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on February 4, 2004.

By:

*Michele Morrow*  
Michele Morrow

TRANSMITTAL DOCUMENT

Commissioner for Patents  
P.O. Box 1450  
Alexandria, VA 22313-1450

Sir:  
ENCLOSED HEREWITH:

- Appellant's Brief (in triplicate) (37 C.F.R. 1.192); and
- Our return postcard.

A fee of \$330.00 is required for filing an Appellant's Brief. Please charge this fee to IBM Deposit Account No. 09-0447. No additional fees are believed to be necessary. If, however, any additional fees are required, I authorize the Commissioner to charge these fees which may be required to IBM Corporation Deposit Account No. 09-0447. No extension of time is believed to be necessary. If, however, an extension of time is required, the extension is requested, and I authorize the Commissioner to charge any fees for this extension to IBM Corporation Deposit Account No. 09-0447.

Respectfully submitted,

*Duke W. Yee*  
Duke W. Yee

Registration No. 34,285

CARSTENS, YEE & CAHOON, LLP

P.O. Box 802334

Dallas, Texas 75380

(972) 367-2001

ATTORNEY FOR APPLICANTS

RECEIVED  
FEB 12 2004  
Technology Center 2100



Docket No. AT9-98-920

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

SS  
#6  
2/20/04

In re application of: **Cooper et. al.**

§

Serial No. 09/306,189

§

Group Art Unit: 2176

§

Filed: May 6, 1999

§

Examiner: **Yuan, Almari Romero**

§

For: **Method and Apparatus for  
Converting Programs and Source  
Code Files Written in a Programming  
Language to Equivalent Markup  
Language Files**

§

§

§

§

§

§

RECEIVED

FEB 12 2004

Technology Center 2100

Commissioner for Patents  
P.O. Box 1450  
Alexandria, VA 22313-1450

ATTENTION: Board of Patent Appeals  
and Interferences

Certificate of Mailing Under 37 C.F.R. § 1.8(a)

I hereby certify this correspondence is being deposited with the United States Postal Service as First Class mail in an envelope addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on February 4, 2004.

By:

Michele Morrow  
Michele Morrow

**APPELLANT'S BRIEF (37 C.F.R. 1.192)**

This brief is in furtherance of the Notice of Appeal, filed in this case on December 11, 2003.

The fees required under § 1.17(c), and any required petition for extension of time for filing this brief and fees therefore, are dealt with in the accompanying TRANSMITTAL OF APPEAL BRIEF.

This brief is transmitted in triplicate. (37 C.F.R. 1.192(a))

### **REAL PARTIES IN INTEREST**

The real party in interest in this appeal is the following party: International Business Machines, Inc.

### **RELATED APPEALS AND INTERFERENCES**

With respect to other appeals or interferences that will directly affect, or be directly affected by, or have a bearing on the Board's decision in the pending appeal, there are no such appeals or interferences.

### **STATUS OF CLAIMS**

#### **A. TOTAL NUMBER OF CLAIMS IN APPLICATION**

Claims in the application are: 6-11, 17-22, and 25

#### **B. STATUS OF ALL THE CLAIMS IN APPLICATION**

1. Claims canceled: NONE
2. Claims withdrawn from consideration but not canceled: NONE
3. Claims pending: 6-11, 17-22, and 25
4. Claims allowed: NONE
5. Claims rejected: 6-11, 17-22, and 25

#### **C. CLAIMS ON APPEAL**

The claims on appeal are: 6-11, 17-22, and 25

### **STATUS OF AMENDMENTS**

No amendments to the claims after final rejection have been made.

## **SUMMARY OF INVENTION**

The present invention provides a method and apparatus for converting programs and source code files written in a programming language to equivalent markup language files. The conversion may be accomplished by a static process or a dynamic process. In a static process, a document type definition file for a markup language is parsed to select an element based on an association between the element and an identifier of a routine in the source code statement. The selected element is written to the markup file. In a dynamic process, the application program is executed to generate the markup language file that corresponds to the source code file of the program. The application program is executed and a document type definition file for a markup language is provided as input. An element defined in the document type definition file is selected based on a routine called by the application program and the selected element is written to a markup language file.

## **ISSUES**

The only issue on appeal is whether claims 6-11, 17-22, and 25 are obvious in view of Meltzer (U.S. Patent No. 6,226,675 B1).

## **GROUPING OF CLAIMS**

The claims do not stand or fall together for the reasons set forth hereafter in Appellants' arguments. The claims stand or fall according to the following grouping of claims:

Group I: claims 6, 9, 10, 11, 17, 20, 21, 22 and 25;

Group II: claims 7 and 18; and

Group III: claims 8 and 19.

## **ARGUMENT**

### **I. 35 U.S.C. 103, Alleged Obviousness, Claims 6-11, 17-22, and 25**

The Final Office Action rejects claims 6-11, 17-22, and 25 under 35 U.S.C. 103(a) over Meltzer

(U.S. Patent No. 6,226,675 B1). This rejection is respectfully traversed.

With regard to independent claim 6, the Final Office Action states:

Regarding independent claims 6, 17, and 25, Meltzer discloses:

A method, data processing system, and computer program product on a computer readable medium of dynamically translating an application program into a markup language file (Meltzer on col. 30, lines 55-61: teaches transforming JAVA into XML), comprising:

- executing an application program (Meltzer on col. 23, lines 17-60: teaches the objects would be transformed into format required by the receiving application);

- parsing a document type definition file for a markup language (Meltzer on col. 23, lines 38-60: teaches parsing a document to retrieve DTD (document type));

- during execution of said program application (on col. 23, lines 17-60: teaches running listeners as JAVA functions); selecting an element defined in the document type definition file based on a routine called by the application program (Meltzer on col. 23, lines 17-60: teaches element retrieved from XML DTD; on col. 5, lines 1-9: teaches particular fields of a document are translated into JAVA objects; on col. 30, lines 55-61: teaches JAVA beans correspond to the logical structures in the DTD for transforming from XML to JAVA and from JAVA to XML); and

- writing the selected element to a markup language file (Meltzer on col. 23, lines 38-60: teaches producing an output by received XML element).

Meltzer on col. 23, lines 17-37 and col. 25, line 52 – col. 26, line 9 teaches a listener as a JAVA program (as routine called by the application program) to listen for events to exploits the JAVA beans API for transforming the object (beans) into a format such as XML; each object becomes an element within that element each embedded method also becomes an element whose content is the value returned by invoking the method.

It would have been obvious to a person of ordinary skill in the art at the time the invention was made to have modified Meltzer to provide a JAVA listener to listen for events to exploit JAVA beans API incorporated as routine called by the application program, in order to enable diverse and flexible implementations of transaction processes of filtering and responding to incoming documents.

Final Office Action dated September 23, 2003, pages 2-3.

Independent claim 6, which is representative of independent claims 17 and 25 with regard to similarly recited subject matter, reads as follows:

6. A method of dynamically translating an application program into a markup language file, the method comprising the computer-implemented steps of:

- executing said application program;

- parsing a document type definition file for a markup language;

- during execution of said application program, selecting an element defined in the

document type definition file based on a routine called by said application program; and writing the selected element to a markup language file to form a translation.  
(emphasis added)

Meltzer teaches a method of exchanging self-defining electronic documents, such as XML based documents, between trading partners without custom integration. The definitions of the electronic business documents, called business interface definitions, are posted on the Internet, or communicated to members of the network. The business interface definition tells potential trading partners the services the company offers and the documents to use when communicating with such services. Participants are programmed by the composition of the input and output documents, coupled with interpretation information in a common business library, to handle the transaction in a way, which closely parallels the way in which paper businesses operate.

Meltzer does not teach that during execution of an application program, an element defined in a document type definition file is selected based on a routine called by the application program, as recited in claim 6. The Final Office Action alleges that Meltzer teaches these features at col. 23, lines 17-37 and col. 25, line 52 to col. 26, line 9, which read as follows:

According to the present invention the applications that the listeners run need not be native XML functions, or native functions which match the format of the incoming document. Rather, these listeners may be Java functions, if the transaction process front end 304 is a Java interface, or may be functions which run according to a unique transaction processing architecture. In these cases, the objects would be transformed into the format required by the receiving application. When the application of the listener finishes, its output is then transformed back into the format of a document specified by the business definition in the module 303. Thus, the translator 302 is coupled to the network interface 300 directly for supplying the composed documents as outputs.

The listeners coupled to the transaction processing front end may include listeners for input documents, listeners for specific elements of the input documents, and listeners for attributes stored in particular elements of the input document. This enables diverse and flexible implementations of transaction processes at the participant nodes for filtering and responding to incoming documents.

(Column 23, lines 17-37, Meltzer)

The idea of an object which walks a tree and generates a stream of events can be generalized beyond the tree of DOM objects, to any tree of objects which can be queried. Thus a Java walker 512 may be an application which walks a tree of Java bean components 513. The walker walks over all the publicly accessible fields and methods. The walker keeps track of the objects it has already visited to ensure that it doesn't go into an endless cycle. Java events 514 are the type of events generated by the Java walker 512. This currently includes most of the kinds of information one can derive from an object. This is the Java equivalent of ESIS and allows the same programming approach applied to XML to be applied to Java objects generally,

although particularly to Java beans.

The Java to XML event generator 515 constitutes a Java listener and a Java event generator. It receives the stream of events 514 from the Java walker 512 and translates selected ones to present a Java object as an XML document. In the one preferred embodiment, the event generator 515 exploits the Java beans API. Each object seen becomes an element, with the element name the same as the class name. Within that element, each embedded method also becomes an element whose content is the value returned by invoking the method. If it is an object or an array of objects, then these are walked in turn.  
(Column 25, line 52 – Column 26, line 9, Meltzer)

In the first section, Meltzer teaches a listener that is coupled to a transaction processing front end to listen for an input document, specific elements of an input document, or attributes stored in particular elements of the input document. The input document, in the case of Meltzer, is an input XML document. Therefore, Meltzer merely teaches using a listener to translate elements in an input XML document to Java. Meltzer does not teach that during execution of an application program, an element is selected based on a routine called by the application program, as recited in claim 6. In the present invention, an element is selected based on a routine called by the application program, for example, a Java application, and the selected element is written to a markup language document, for example, an output XML document. Hence, the element of the present invention is selected to translate a routine called in an application to elements of an output markup language document. To the contrary, Meltzer teaches the opposite, that is, using the listener for translating XML elements to Java.

In the second section, Meltzer teaches a Java to XML event generator that receives a stream of events and translates selected ones to present a Java object as an XML document. The “selected ones” as referred to in Meltzer, are selected Java events generated by the Java walker as the Java walker walks the DOM object. The selected Java events are not elements selected based on a routine called by the application program, as recited in claim 6. The selected Java events are not necessary routines called by the application program. As Meltzer later teaches, each object that the Java walker walks becomes an element. Within each element, each embedded method becomes an element whose content is the value returned by invoking the method. Meltzer explicitly teaches that each embedded method becomes an element, whether the method is called by the application program or not. Therefore, Meltzer selects an element based on the criteria that a particular routine is embedded within an object, as opposed to a routine called by the application program during the execution of the application program as recited in claim 6.

It is clear that with Meltzer, every method in the object will be selected as an element. This is contrary to the presently claimed invention, which recites that an element is selected based on a routine called by the application program. Only an element corresponding to a routine called by the application program is selected. Thus, the present invention has an advantage over Meltzer in that the present invention includes the ability to select elements based on a routine called by the application program. Meltzer does not teach or suggest, in any section of the reference, the ability to select an element based on a routine called by the application, as recited in claim 6. To the contrary, because Meltzer performs operations on non-executed code, all of the methods of an object are made elements. Therefore, a person of ordinary skill in the art would not be led to modify Meltzer's listener to reach the presently claimed invention simply because Meltzer does not teach or suggest selecting an element based on a routine called by an application.

Meltzer actually teaches away from the presently claimed invention because Meltzer explicitly teaches each embedded method within an object becomes an element, as opposed to selecting an element based on a routine called by the application. Absent the Examiner pointing out some teaching or incentive to implement Meltzer in this manner, one of ordinary skill in the art would not be led to modify Meltzer to reach the presently claimed invention when the reference is examined as a whole. Absent some teaching, suggestion, or incentive to modify Meltzer in this way, the presently claimed invention can be reached only through an improper use of hindsight using the Applicants' disclosure as a template to make the necessary changes to reach the presently claimed invention.

In view of the above, Appellants respectfully submit that Meltzer does not teach or suggest the features recited in claim 6. The other independent claims 17 and 25 recite similar subject matter also not taught by Meltzer. Thus, Appellants respectfully request withdrawal of the rejection of claims 6, 17 and 25. At least by virtue of their dependency on claims 6 and 17 respectively, Meltzer does not teach or suggest the features set forth in dependent claims 7-11 and 18-22. Accordingly, Appellants respectfully request withdrawal of the rejection of claims 7-11 and 18-22 under U.S.C. 103(a).

In addition, Meltzer does not teach or suggest any of the specific features set forth in the dependent claims 7-11 and 18-22. With regard to dependent claim 7, which is representative of claim 18 with regard to similarly recited subject matter, Meltzer does not teach that an element comprises an attribute list corresponding to parameters for a routine that is executed. The Final



Office Action alleges that Meltzer teaches these features at column 76, lines 33-67, which reads as follows:

A compiler takes the purchase order definition and generates several different target forms. All of these target forms can be derived through “tree to tree” transformations of the original specification. The most important for this example are:

- (a) the XML DTD for the purchase order
- (b) a Java Bean that encapsulates the data structures for a purchase order (the Java classes, arguments, datatypes, methods, exception structures are created that correspond to information in the Schema definition of the purchase order).
- (c) A “marshaling” program that converts purchase orders that conform to the Purchase Order DTD into a Purchase Order Java Bean or loads them into a database, or creates HTML (or an XSL style sheet) for displaying purchase orders in a browser.
- (d) An “unmarshaling” program that extracts the data values from Purchase Order Java Beans and converts them into an XML document that conforms to the Purchase Order DTD.

Now back to the scenario. A purchasing application generates a Purchase Order that conforms to the DTD specified as the service interface for a supplier who accepts purchase orders.

The parser uses the purchase order DTD to decompose the purchase order instance into a stream of information about the elements and attribute values it contains. These “property sets” are then transformed into corresponding Java event objects by wrapping them with Java code. This transformation in effect treats the pieces of marked-up XML document as instructions in a customer programming language whose grammar is defined by the DTD. These Java events can now be processed by the marshaling applications generated by the compiler to “load” Java bean data structures.

(Column 76, lines 33-67, Meltzer)

In the above section, Meltzer teaches an unmarshaling program that extracts data values from a purchase order Java bean and converts them into an XML document that conforms to the Purchase Order. However, Meltzer does not teach an element that comprises an attribute list corresponding to parameters for the routine. While Meltzer teaches a Java bean that encapsulates data structures of a purchase order that include arguments, Meltzer does not teach an element that comprises an attribute list of these arguments (parameters for the routine). To the contrary, Meltzer teaches extracting only data values from the Java bean to convert to an XML document. When converting from XML to Java using Meltzer’s marshaling program, the parser parses the purchase order XML document according to the document type definition to retrieve elements and attributes of the XML document. However, when converting from Java to XML, Meltzer’s unmarshaling program only calls for extracting data values from the Java bean, not parameters for the routine, as recited in claim 7.

In addition, at column 25, line 52 to column 26, line 9, which is reproduced above, Meltzer teaches that each embedded method becomes an element whose content is the value returned by invoking the method. Meltzer does not teach an element comprising an attribute list of parameters for the routine, since the parameters for the routine are inputs to the routine. Rather, Meltzer teaches an element comprising a value returned by invoking the method. The value returned is output of the routine. In Java, as known by a person of ordinary skill in the art, “parameters” for a routine carries a meaning of input parameters or input arguments of a routine. Thus, “parameters” for a routine are commonly associated with inputs to a routine. To the contrary, a “returned value” is commonly known as an output value from invoking a routine. In the case of Java, only one returned value is normally associated for a routine, which represents an output of the routine. Therefore, Meltzer does not teach an element comprising an attribute list of “parameters” for the routine, since Meltzer only teaches an element whose content is output of a routine, not parameters (inputs) for the routine.

With regard to dependent claim 8, which is representative of claim 19 with regard to similarly recited subject matter, Meltzer does not teach that a selected element that is written to a markup language file comprises an attribute list corresponding to values for parameters passed to the routine. The Final Office Action alleges that Meltzer teaches these features at column 76, lines 33-67, which is reproduced above. However, as described above, Meltzer teaches that only data values are written to the XML document by the unmarshaling program. The data values of Meltzer are values returned by invoking the routine, not values for the parameters passed to the routine. Meltzer is only interested in placing returned data as a result of invoking a routine into the XML document, Meltzer is not interested in placing values passed to the parameters for the routine in the XML document, as recited in claim 8 of the presently claimed invention. The data values returned by invoking a routine are outputs of the routine. To the contrary, values passed to the routine for the parameters are inputs to the routine. Therefore, one of ordinary skill in the art would not be led to modify Meltzer’s teaching to reach the presently claimed invention, because Meltzer does not teach the selected element written to the markup language file comprises an attribute list corresponding to values for the parameters passed to the routine as recited in claim 8.

Thus, in addition to being dependent on independent claim 6, the dependent claims 7, 8, 18, and 19 are also allowable by virtue of their specific recited features. Accordingly, the rejections of claims 8, 18 and 19 should be withdrawn.

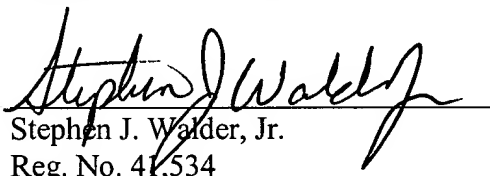
## **II. Summary**

In summary, Meltzer fails to teach or suggest the specific features of the present invention. There is nothing in Meltzer that teaches or suggests selecting an element defined in the document type definition file based on a routine called by an application program. There is nothing in Meltzer that teaches or suggests that the element comprises an attribute list corresponding to parameters for the routine or that a selected element written to a markup language file comprises an attribute list corresponding to values for the parameters passed to the routine.

## **III. Conclusion**

In view of the above, Appellants respectfully submit that claims 6-11, 17-22, and 25 define over the prior art of record, Meltzer. Appellants therefore respectfully request the Board of Patent Appeals and Interferences to overturn the rejection of claims 6-11, 17-22, and 25 under 35 U.S.C. 103(a).

Respectfully submitted,

  
Stephen J. Walder, Jr.  
Reg. No. 41,534  
Carstens, Yee & Cahoon, LLP  
PO Box 802334  
Dallas, TX 75380  
(972) 367-2001

## **APPENDIX OF CLAIMS**

The text of the claims involved in the appeal are:

6. A method of dynamically translating an application program into a markup language file, the method comprising the computer-implemented steps of:

executing said application program;

parsing a document type definition file for a markup language;

during execution of said application program, selecting an element defined in the document type definition file based on a routine called by said application program; and

writing the selected element to a markup language file to form a translation.

7. The method of claim 6 wherein the element comprises an attribute list corresponding to parameters for the routine.

8. The method of claim 6 wherein the selected element written to the markup language file comprises an attribute list corresponding to values for the parameters passed to the routine.

9. The method of claim 6 wherein the application program is written in Java programming language.

10. The method of claim 9 wherein the routine is an extended class method.

11. The method of claim 9 wherein the routine is a Graphics class method.

17. A data processing system for dynamically translating an application program into a markup language file, the data processing system comprising:
- executing means for executing an application program;
  - parsing means for parsing a document type definition file for a markup language;
  - selecting means for selecting an element defined in the document type definition file based on a routine called by the application program; and
  - writing means for writing the selected element to a markup language file to form a translation.
18. The data processing system of claim 17 wherein the element comprises an attribute list of parameters for the routine.
19. The data processing system of claim 17 wherein the selected element written to the markup language file comprises an attribute list of values for the parameters passed to the routine.
20. The data processing system of claim 17 wherein the application program is written in Java programming language.
21. The data processing system of claim 20 wherein the routine is an extended class method.
22. The data processing system of claim 20 wherein the routine is a Graphics class method.

25. A computer program product on a computer readable medium for use in a data processing system for dynamically translating an application program into a markup language file, the computer program product comprising:

- first instructions for executing an application program;
- second instructions for parsing a document type definition file for a markup language;
- third instructions for selecting an element defined in the document type definition file based on a routine called by the application program; and
- fourth instructions for writing the selected element to a markup language file to form a translation.